Laboratotio Arquitectura de Computadores

Santiago Ramirez Arenas
Docente: José Alfredo Jaramillo Villegas
Universidad Tecnológica de Pereira

12 de noviembre de 2021

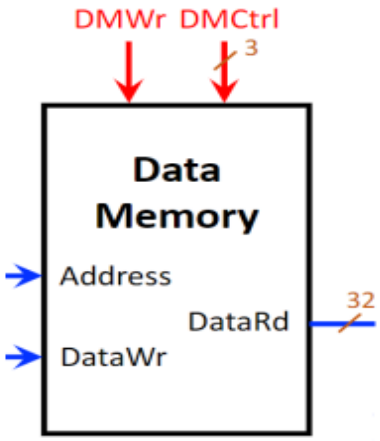
Memoria de datos.

Se define como una matriz en la cual se puede almacenar y leer los datos que van a ser procesados. Este módulo del procesador se encarga de almacenar todos los datos que van a perdurar y, a su vez, recuperar los datos solicitados. El módulo cuenta con los siguientes registros de entrada y salida (cada uno con su respectivo número de bits):

o Escritura: La dirección de ingreso para almacenar (Address), el valor que se quiere almacenar (DataWr) y la bandera que activa la escritura (DMWr).

o Lectura: Tipo de lectura DMCtrl (Byte, Halfword, Word, Unsigned Byte, Unsigned Halfword), La dirección de ingreso para leer la memoria (Address).

o Nota: Address funciona para los casos (lectura y escritura)



DMCtrl	Tipo de dato
000	B
001	H
010	W
100	B (U)
101	H (U)

Implementación Memoria de datos

```
// Code your design here

module DataMemory(
       input logic [31:0] Address,
  input logic [31:0] DataWr,
       input logic DMWr,
       input logic DMRd,
       output logic [31:0] DataRd);

  logic [31:0] Matrix [31:0];

        initial
               $readmemh("registers.txt", Matrix);
                      // Reading
                  always@(*)
                      begin
                         if (DMRd== 1)
                               DataRd = Matrix[Address];
                      end
                       //Writing
                    always@(*)
                       begin
                         if (DMWr == 1)
                           Matrix [Address]= DataWr;
                       end
                   endmodule
```

Registers.txt

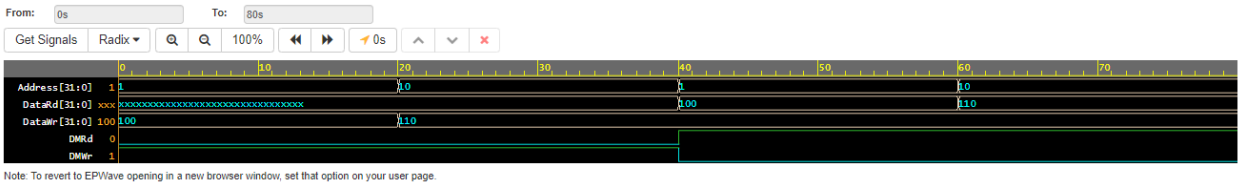
```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

Testbench Memoria de Datos

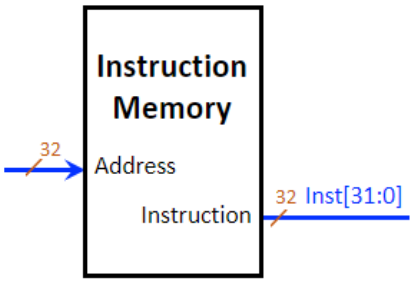
```
// Code your testbench here
// or browse Examples
module Testbench;
  logic [31:0] Address = 0;
  logic [31:0] DataWr = 0;
  logic DMWr = 0;
  logic DMRd = 0;
  logic [31:0] DataRd;

  DataMemory datamemory (Address, DataWr, DMWr, DMRd, DataRd);

      initial
            begin
              $dumpfile ("dump.vcd");
              $dumpvars (1);
                    DMWr = 1; Address = 1; DataWr = 4;
                    #20 DMWr = 1; Address = 2; DataWr = 6;
                    #20 DMWr = 0; DMRd = 1; Address = 1;
                    #20 DMWr = 0; DMRd = 1; Address = 2;
                    #20
              $finish ();
            end
      endmodule
```

Simulación Memoria de Datos



Memoria de instrucciones.
Se puede definir como una matriz en la cual se almacenan y se leen las instrucciones que se van a ejecutar.

Este módulo del procesador recibe un Address el cual es la dirección de la instrucción que se quiere recuperar y una salida Instruction la cual es la instrucción que fue recuperada de la matriz.



Diseño Memoria de Instrucciones

```
 // Code your design here
module InstuctionMemory(
  input logic [31:0]Address,
  output logic [31:0] Instruction);

  logic [31:0] Matrix [31:0];
        initial
         $readmemh ("registers.txt",Matrix);

   assign Instruction = Matrix [Address];
endmodule
```

registers.txt

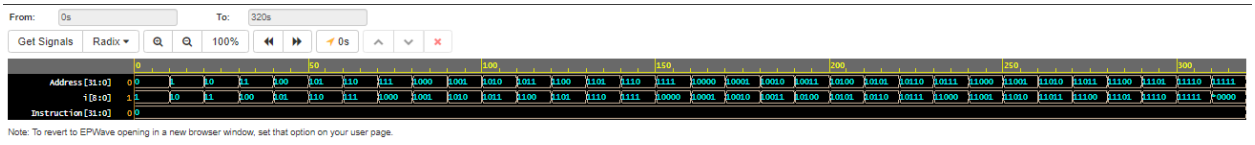
```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

```
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
00000000
```

Testbench Mempria de Instrucciones

```
// Code your testbench here
// or browse Examples
     module Testbench;
       logic [31:0] Address = 0;
       logic [31:0] Instruction;

       InstuctionMemory instmemory (Address, Instruction);
       logic [8:0] i= 1;
              initial
                   begin
                     $dumpfile ("dump.vcd");
                     $dumpvars (1);
                     for (i = 1; i <= 32; i=i+1)
                            begin
                                   #10
                                   Address = i;
                            end
                     $finish();
                   end
             endmodule
```

Simualción Memoria de Isntrucciones



Función factorial

Crear una función en C o C++ de la operación matemática factorial. Después llevar esta función a lenguaje ensamblador y a lenguaje de maquina (Binario y hexadecimal). Con el resultado de las instrucciones en lenguaje de máquina, se debe inicializar la memoria de instrucciones en el primer punto

Función Factorial

Función en C++

```
int multi(int x, int y)
{
 int Acc = 0;
 for(int i = 0;i < y;i++)
 Acc += x;
 return Acc;
}
 int factorial(int num)
{
 int fact = 1;
 int num += 1;
 for(int i = 1;i < num;i++)
 fact = multi(fact,i);
 return fact;
 }
```

Función en lenguaje ensamblador

Prototipo de la función multi:

- o 10 ->x ->x8
- o x11 ->y ->x9
- o Acc ->x18
- o i ->x19
- o multi(x,y) ->x10
- o Prototipo de la función factorial:
- o x10 ->num ->x8
- o i ->x9
- o fact ->x18
- o factorial(num) ->x10
- o Llamado a la función factorial:
- o f = factorial(num)
- o num ->x8
- o f ->x9

Lenguaje ensamblador			
Address	**Label**	**Instruction**	**Description**
0x0…4200	multi:	addi sp,sp,-16	**Reservamos espacio en la pila**
0x0…4204		sw x8,12(sp)	**Guardamos en los espacios reservados anteriormente**
0x0…4208		sw x9,8(sp)	
0x0…420C		sw x18,4(sp)	
0x0…4210		sw x19,0(sp)	
0x0…4214		addi x8,x10,0	**Movemos los argumentos**
0x0…4218		addi x9,x11,0	
0x0…421C		addi x18,x0,0	**Acc = 0**
0x0…4220		addi x19,x0,0	**i = 0**
0x0…4224	for0:	bge x19,x9,endfor0	**Cuando (i >= y) salir del for**
0x0…4228		add x18,x18,x8	**Acc += x**
0x0…422C		addi x19,x19,1	**i++**
0x0…4230		beq x0,x0,for0	
0x0…4234	endfor0:	addi x10,x18,0	**return Acc**
0x0…4238		lw x8,12(sp)	**Recuperar de la pila los valores en los que recibí los registros.**
0x0…423C		lw x9,8(sp)	
0x0…4240		lw x18,4(sp)	
0x0…4244		lw x19,0(sp)	
0x0…4248		addi sp,sp,16	**Liberar el espacio reservado**
0x0…424C		jalr x0,0(x1)	**Retorno**
…			
0x0…4300		addi x10,x8,0	
0x0…4304		jal x1,factorial	
0x0…4308		addi x9,x10,0	
…			
0x0…4400	factorial:	addi sp,sp,-16	**Reservar espacio en la pila.**
0x0…4404		sw x8,12(sp)	**Guardar en el espacio reservado en la pila los registros.**
0x0…4408		sw x9,8(sp)	
0x0…440C		sw x18,4(sp)	
0x0…4410		sw x1,0(sp)	
0x0…4414		addi x8,x10,0	**Mover los argumentos**
0x0…4418		addi x18,x0,1	**fact = 1**
0x0…441C		addi x8,x8,1	**num += 1**
0x0…4420		addi x9,x0,1	**i = 1**
0x0…4424	for0:	bge x9,x8,endfor0	**Si (i >= num) salir del for**
0x0…4428		addi x10,x18,0	**fact = multi(fact,i)**
0x0…442C		addi x11,x9,0	
0x0…4430		jal x1,multi	
0x0…4434		addi x18,x10,0	
0x0…4438		addi x9,x9,1	**i++**
0x0…443C		beq x0,x0,for0	

0x0…4440	endfor0:	addi x10,x18,0	**return fact**
0x0…4444		lw x8,12(sp)	**Recuperar de la pila los valores en los que recibí los registros.**
0x0…4448		lw x9,8(sp)	
0x0…444C		lw x18,4(sp)	
0x0…4450		lw x1,0(sp)	
0x0…4454		addi sp,sp,16	**Liberar el espacio reservado**
0x0…4458		jalr x0,0(x1)	**Retorno**

Conversión lenguaje de Máquina Función Multi

Addi sp,so,-16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
F				F				0				1				0				1				1				3			

Hexadecimal 0Xff010113

Sw x8, 12 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1
0				0			8					1				2				6				2				3			

Hexadecimal 0x00812623

Sw x9, 8 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1
0				0			9					1				2				4				2				3			

Hexadecimal 0x00912423

Sw x18, 4 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
0				1			2					1				2				2				2				3			

Hexadecimal 0x01212223

Sw x19, 0 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	1	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1
9				1			3					1				2				0				2				3			

Hexadecimal 0x1312023

Addi x8, x10, 0

Imm [11:5]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			0					5				0				4				1				3			

Hexadecimal 0x00050413

Addi x18,x0,0

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0			0					5				8				4				9				3			

Hexadecimal 0x0058493

Addi x19, x0,0

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1
0				0			0					0				0				9				9				3			

Hexadecimal 0x00000913

Bge x19,x9, end for ()

Imm [12,10;5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	1	0	0	1	1	1	0	1	1	0	0	0	0	1	1	0	0	0	1	1
0				0			9					9					D			8					6			3			

Hexadecimal 0x0099D863

Add, x18,x18,x8

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	1	0	0	1	1
0				0			8					9					0			9					3			3			

Hexadecimal 0x00890933

Addi x19, x19, 1

Imm [11:0]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	1	0	0	1	1	0	0	0	1	0	0	1	1	0	0	1	0	0	1	1
0				0			1					9					8			9					3			3			

Hexadecimal 0x00198993

Beq x0, x0, for 0

Imm [12,10;15]							Rs2					Rs1					Funct3			Imm [4:0,11]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	1	1	1	0	0	0	1	1
F				E			0					0					0			A					E			3			

Hexadecimal 0XFE000AE3

Addi x10, x18,0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0			0					9					0			5					2			3			

Hexadecimal 0x00090513

lw x8, 12 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1
0				0			C					1					2			4					0			3			

Hexadecimal 0x00C12403

lw x9,8(sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	0	0	0	0	0	1	1
0				0			8					1					2			4					8			3			

Hexadecimal 0x00812483

lw x18, 4 (2p)

Imm [11:0]							Rs2					Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1
0				0			4					1					2			9					0			3			

Hexadecimal 0x00412903

Iw x19, 0 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	1	0	0	1	1	0	0	0	0	0	1	1
0				0				0				1					2			9					8					3	

Hexadecimal 0x00012983

Addi sp, sp, 16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
0				1				0				1					0			1					1					3	

Hexadecimal 0x01010113

Jarlr x0, 0 (x1)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1
0				0				0				0					8			0					6					7	

Hexadecimal 0x00008067

Addi x10, x8,0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				4					0			5					1					3	

Hexadecimal 0x00040513

Jal x1, factorial

Imm [20,10: 1, 11,19:12]																				rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	0	1	1	1	1
0				F				C				0					0			0					E					F	

Hexadecimal 0x0FC000EF

Addi x9, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0				0				5					0			4					9					3	

Hexadecimal 0x0050493

Addi sp, sp, -16

Imm [11:5]												Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
F				F				0				1					0			1					1					3	

Hexadecimal 0xFF010113

Sw x8, 12 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0	1	0	0	1	1	0	0	0	1	0	0	0	1	1
0				0				8				1					2			6					2					3	

Hexadecimal 0x00812623

Sw x9, 8 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	0	0	1	0	0	0	0	1	0	0	0	1	1
0				0			9					1					2			4					2			3			

Hexadecimal 0x00912423

Sw x18, 4 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	0	1	0	0	0	1	1
0				1			2					1					2			2					2			3			

Hexadecimal 0x01212223

W2 x1, 0 (sp)

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	1
0				0			1					1					2			0					2			3			

Hexadecimal 0x00112023

Addi x8, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			0					5					0			4					1			3			

Hexadecimal 0x00050413

Addi x18, x0, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1
0				0			1					0					0			9					1			3			

Hexadecimal 0x00100913

Addi x8 , x8, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	1
0				0			1					4					0			4					1			3			

Hexadecimal 0x00140413

Addi x9, x0, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0			1					0								4					9			3			

Hexadecimal 0x00100493

Bge x9, x8, endfor 0

Imm [11:5]							Rs2					Rs1					Funct3			Imm [4:0]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	1	0	0	1	1	0	1	1	1	1	0	0	1	1	0	0	0	1	1
0				0			8					4					D			E					6			3			

Hexadecimal 0x0084DE63

Addi x10, x18, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				9				0				5				1				3			

Hexadecimal 0x00090513

Addi x11, x9, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	1	1	0	0	1	0	0	1	1
0				0				0				4				8				5				9				3			

Hexadecimal 0x00048593

Jal x, multi

Imm [11:5]																				rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	0	1	1	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1	0	0	0	0	1	1	1	0	1	1	1	
D				D				1				F				F				0				E				F			

0xDDIFFOEF

Addi x18, x10, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	0	0	1	0	0	1	0	0	0	1	0	0	1	1
0				0				0				5				0				9				1				3			

Hexadecimal 0x00050913

Addi x9, x9, 1

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0	0	0	1	0	0	1	0	0	1	0	0	1	1
0				0				1				4				8				4				9				3			

Hexadecimal 0x00148493

Beq x0, x0,for 0

Imm [11:0]							Rs2					Rs1					Funct3			Imm [4:0,11]					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
1	1	1	1	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	1	1	0	0	0	1	1
F				E				0				0				0				4				E				3			

Hexadecimal 0XFE0004E3

Add x10, x18, 0

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	0	0	1	1
0				0				0				9				0				5				1				3			

Hexadecimal 0x00090513

Iw x8, 12 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	1	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	1

0	0	C	1	2	4	0	3

Hexadecimal 0x00C12403

lw x9, 8 (sp)

Imm [11:0]												Rs1					Funct3			Ird					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	1	0	1	0	1	0	0	1	0	0	0	0	0	1	1
0				0				8				1					2			4					8					3	

Hexadecimal 0x00812483

lw x18, 4 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	1	0	1	0	0	1	0	0	0	0	0	0	1	1
0				0				4				1					2			9					0					3	

Hexadecimal 0x00812623

lw x1, 0 (sp)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	1	0	0	0	0	1	0	0	0	0	0	0	1	1
0				0				0				1					2			0					8					3	

Hexadecimal 0x00812623

Addi sp, sp, 16

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	1	0	0	1	1
0				1				0				1					0			1					1					3	

Hexadecimal 0x01010113

Jarl x0, 0 (x1)

Imm [11:0]												Rs1					Funct3			rd					Opcode						
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	29	20	21	22	23	24	25	26	27	28	29	30	31	32
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	1	0	0	1	1	1
0				0				0				8					0			6					7						

Hexadecimal 0x00008067

Diseño con instrucciones

```
module InstuctionMemory(
  input logic [31:0]Address,
  output logic [31:0] Instruction);

  logic [31:0] Matrix [45:0];

       initial
         $readmemh ("registers.txt",Matrix);

  assign Instruction = Matrix [Address];
endmodule
```

Instrucciones

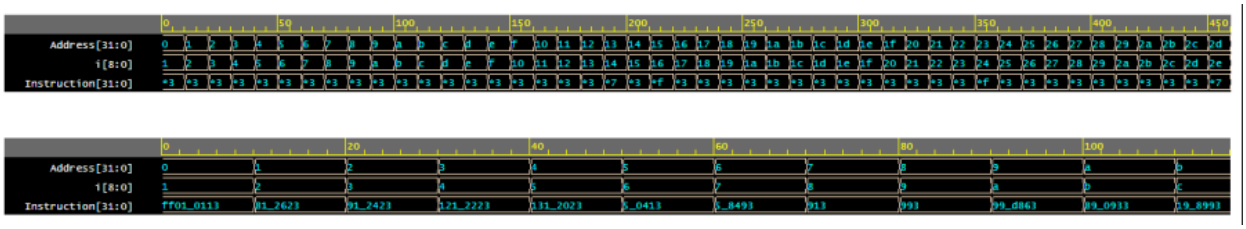
```
FF010113
00812623
00912423
01212223
01312023
00050413
00000913
00000993
00000993
0099D863
00890933
00198993
FE000AE3
00090513
00C12403
00812483
00412903
00012983
01010113
00008067
00040513
0FC000EF
00050493
FF010113
00812623
00912423
01212223
00112023
00050413
00100913
00140413
00100493
0084DE63
00090513
00048593
DD1FF0EF
00050913
00148493
FE0004E3
```

```
00090513
00C12403
00812483
00412903
00012083
00012083
01010113
00008067
```

Testbench con Instrucciones

```
// Code your testbench here
// or browse Examples
     module Testebench;
       logic [31:0] Address = 0;
       logic [31:0] Instruction;

       InstuctionMemory instmemory (Address, Instruction);
       logic [8:0] i= 1;
              initial
                   begin
          $dumpfile ("dump.vcd");
          $dumpvars (1);
             for (i = 1; i <= 23; i=i+1)
                  begin
                     #10
                     Address = i;
                end
             $finish();
           end
  endmodule
```

Simulación con instrucciones de la función Factorial



Procedimiento;
Para el diseño de la memoria de datos, se tomó como referencia la ilustación gráfica en dodnde se evidencia que cada señal activa una acción a realizar ya sea escirtura o lectura según los valores asigandos, además se realizó un módulo externo para almacenar la información de la misma forma en la que lo hace una memoria.

Con respecto a la conversión de las instrucciones de lenguaje de lenguaje C, a ensamblador y posteriormente a lenguaje de máquina, se realizó en formato de tabla para una mejor compresión y un mayor orden, dando a conocer la conversión de cada instrucción.
Problemas encontrados:
Es un poco complicado realizar la articulación entre el diseño y el módulo de testo para

guardar los cambios realizados en la memoria, pero con la documentación que se encuentra en internet se pudo solucionar.

- Interpretación de los resultados de la simulación.

En las simulaciones se observa un correcto funcionamiento, dado que cuando los activadores de escritura y lectura están en enable activan los debidos procesos que se observan en la simulación.

En relación a la memoria de instrucciones se observa que los procesos indicados son los correspondientes en relación a su diseñ.

Conclusiones
La meoria de datos es una matriz en donde se almacenan y se leen los datos que posteriormente serán procesados por los demás componente. ára posteriormente en la memoria de instrucciones se leen valga la redundancia las instrucciones que van a ser ejecutadas.

Su set de instrucciones es muy sencillo y a la vez, muy modular. Un núcleo RISC-V puede construirse bien con instrucciones completas (sin necesidad de dividirlas) o contar con extensiones añadidas e instrucciones comprimidas. Precisamente la compresión de las instrucciones es una de las claves de este diseño, lo que le permite ganar en velocidad, siendo a la vez muy eficiente en su uso de memoria.